



2/ AF\$

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, Colorado 80527-2400

PATENT APPLICATION

ATTORNEY DOCKET NO. 200308303-1

IN THE
UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s): Daniel J. Scales

Confirmation No.: 8427

Application No.: 09/757,764

Examiner: Michael J. Yigdall

Filing Date: January 9, 2001

Group Art Unit: 2192

Title: SYSTEM AND METHOD FOR OPTIMIZING OPERATIONS VIA DATAFLOW ANALYSIS

Mail Stop Appeal Brief-Patents
Commissioner For Patents
PO Box 1450
Alexandria, VA 22313-1450

TRANSMITTAL OF APPEAL BRIEF

Transmitted herewith is the Appeal Brief in this application with respect to the Notice of Appeal filed on attached.

The fee for filing this Appeal Brief is (37 CFR 1.17(c)) \$500.00.

(complete (a) or (b) as applicable)

The proceedings herein are for a patent application and the provisions of 37 CFR 1.136(a) apply.

☐ (a) Applicant petitions for an extension of time under 37 CFR 1.136 (fees: 37 CFR 1.17(a)-(d)) for the total number of months checked below:

☐ 1st Month
\$120

☐ 2nd Month
\$450

☐ 3rd Month
\$1020

☐ 4th Month
\$1590

☐ The extension fee has already been filed in this application.

☐ (b) Applicant believes that no extension of time is required. However, this conditional petition is being made to provide for the possibility that applicant has inadvertently overlooked the need for a petition and fee for extension of time.

Please charge to Deposit Account 08-2025 the sum of \$ 500 . At any time during the pendency of this application, please charge any fees required or credit any over payment to Deposit Account 08-2025 pursuant to 37 CFR 1.25. Additionally please charge any fees to Deposit Account 08-2025 under 37 CFR 1.16 through 1.21 inclusive, and any other sections in Title 37 of the Code of Federal Regulations that may regulate fees. A duplicate copy of this sheet is enclosed.

Date: August 4, 2006

I hereby certify that this document is being filed by personal delivery to the Customer Service Window Randolph Building, 401 Dulany Street Alexandria, VA 22314, of the United States Patent & Trademark Office on the date indicated above.

By: [Signature] 48,360
(Attorney Signature and Reg. No.)

Respectfully submitted,

Daniel J. Scales

By

for Patrick C. Keane

Attorney/Agent for Applicant(s)

Reg No. : 32,858

Date :

Telephone : (703) 838-6522



Attorney's Docket No. 200308303-1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of

Daniel J Scales

Application No.: 09/757,764

Filed: January 9, 2001

For: SYSTEM AND METHOD FOR
OPTIMIZING OPERATIONS VIA
DATAFLOW ANALYSIS

Group Art Unit: 2192

Examiner: MICHAEL J YIGDALL

Appeal No.: _____

APPEAL BRIEF

Mail Stop APPEAL BRIEF - PATENTS

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

This appeal is from the decision of the Examiner dated April 11, 2006 finally rejecting claims 1-39, which are reproduced as the Claims Appendix of this brief.

Table of Contents

Table of Contents

	Page
I. Real Party in Interest.....	2
II. Related Appeals and Interferences	2
III. Status of Claims	2
IV. Status of Amendments.....	2
V. Summary Claimed Subject Matter.....	2
VI. Grounds of Rejection to be Reviewed on Appeal.....	4
VII. Argument	5
VIII. Claims Appendix	9
IX. Evidence Appendix	9
X. Related Proceedings Appendix.....	9
XI. Conclusion	11

I. Real Party in Interest

The present application is assigned to Hewlett-Packard Development Company LP. Hewlett-Packard Development Company LP is the real party in interest, and is the assignee of Application No. 09/757,764.

II. Related Appeals and Interferences

The Appellant's legal representative, or assignee, does not know of any other appeal or interferences which will affect or be directly affected by or have bearing on the Board's decision in the pending appeal.

III. Status of Claims

The application contains claims 1-39, all of which are pending and stand finally rejected. This appeal is directed to claims 1-39.

IV. Status of Amendments

No Amendments were filed after the Final Office Action on April 11, 2006.

V. Summary Claimed Subject Matter

As recited in claim 1, a method for modifying serial dependencies in a procedure is disclosed (e.g., pages 2, lines 25 and 26). Such a method is encompassed by claim 1 and includes the steps of building a graph representation of said procedure, said graph representation having an origin and including a unique position, relative to said origin, for each memory operation in said procedure; and designating a location type for each memory operation in said graph representation; each said location type based on a characteristic of said corresponding memory operation (e.g., page 2, lines 26-30). A summary is generated for each memory operation in the graph representation that indicates for each location type used in the procedure the closest preceding memory operation to affect that location type; and a

first memory operation having the same location type as a second memory operation is identified, wherein said first memory operation is positioned closer to said origin than said second memory operation, and said graph representation does not include any additional memory operations of the same location type between the first and second memory operations (e.g., page 3, lines 4-11). Said second memory operation is moved to a new position in said graph representation that is closer to said first memory operation, wherein the moving step includes (i) removing one or more of the serial dependencies in an initial set of serial dependencies that is associated with said second memory operation and (ii) creating a new serial dependency between said first memory operation and said second memory operation (e.g., page 3, lines 18-24).

Claim 13 is directed to a computer program product for use in conjunction with a computer system, the computer program product being capable of modifying serial dependencies in a procedure (e.g., page 8, lines 24-28). The claim 13 computer program product encompasses a computer readable storage medium and a computer program mechanism embedded therein, comprising instructions for building a graph representation of said procedure, said graph representation having an origin and including a unique position, relative to said origin, for each memory operation in said procedure; and instructions for designating a location type for each memory operation in said graph representation; each said location type based on a characteristic of said corresponding memory operation. Also included are instructions for generating a summary for each memory operation in the graph representation that indicates for each location type used in the procedure the closest preceding memory operation to affect that location type; and instructions for identifying a first memory operation having the same location type as a second memory operation; wherein said first memory operation is positioned closer to said origin than said second memory operation, and said graph representation does not include any additional memory operations of the same location type between said first and second memory operations (e.g., page 9, lines 5-19). Further included are instructions for moving the second memory operation to a new position in said graph representation that is closer to said first memory operation, wherein the instructions for moving include (i) instructions for removing one or more of the serial

dependencies in an initial set of serial dependencies that is associated with said second memory operation and (ii) creating a new serial dependency between said first memory operation and said second memory operation (e.g., page 3, lines 18-24; and page 9, lines 19-24).

Claim 25 is directed to a computer system for modifying serial dependencies in a procedure (e.g., page 7, line 29). The claim 25 computer system comprises a memory to store instructions and data; and a processor to execute the instructions stored in the memory (e.g., page 7, lines 29-32). The memory can store instructions for building a graph representation of said procedure, said graph representation having an origin and including a unique position, relative to said origin, for each memory operation in said procedure; instructions for designating a location type for each memory operation in said representation; each said location type based on a characteristic of said corresponding memory operation; instructions for generating a summary for each memory operation in the graph representation that indicates for each location type used in the procedure the closest preceding memory operation to affect that location type; instructions for identifying a first memory operation having the same location type as a second memory operation; wherein said first memory operation is positioned closer to said origin than said second memory operation, and said graph representation does not include any additional memory operations of the same location type between the first and second memory operations; and instructions for moving said second memory operation to a new position in said graph representation that is closer to said first memory operation (e.g., page 9, lines 5-19). The instructions for moving include instructions for removing one or more of the serial dependencies in an initial set of serial dependencies and creating a new serial dependency from the first memory operation to the second memory operation (e.g., page 3, lines 18-24; and page 9, lines 19-24).

VI. Grounds of Rejection to be Reviewed on Appeal

The final Office Action presents the following grounds of rejection to be reviewed on appeal:

1. Claims 1, 2, 5, 7-14, 17, 19-26, 29 and 31-39 stand rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Patent No. 5,758,051 ("the Moreno et

al. patent") in view of U.S. Patent No. 5,787,287 ("the Bharadwaj patent"); U.S. Patent No. 6,077,313 ("the Ruf patent"); and U.S. Patent No. 6,615,403 ("the Muthukumar et al. patent");

2. Claims 3, 4, 15, 16, 27 and 28 stand rejected under 35 U.S.C. 103(a) as being unpatentable over the Moreno patent in view of the Bharadwaj patent, the Ruf patent, the Muthukumar et al. patent, and further in view of U.S. Patent No. 6,016,398 ("the '398 Radigan patent"); and

3. Claims 6, 18 and 30 stand rejected under 35 U.S.C. 103(a) as being unpatentable over the Moreno patent, the Bharadwaj patent, the Ruf patent, the Muthukumar et al. patent, and further in view of U.S. Patent No. 6,151,704 ("the '704 Radigan patent").

VII. Argument

A. The Examiner Has Failed To Establish A Prima Facie Case of Obviousness In Combining The Moreno et al. Patent And The Secondary References To Reject Claims 1, 13 and 25.

The Moreno et al. patent, considered alone, or in combination with the various secondary references relied upon by the Examiner, fails to teach or suggest all features set forth in Appellant's independent claims 1, 13 and 25. Specifically, the Moreno et al. patent fails to teach or suggest at least the argued features of removing a serial dependency in an initial set of serial dependencies and creating a new serial dependency between memory operations as a part of moving a second memory operation to a new position, as variously recited in Appellant's independent claims 1, 13 and 25. In contrast, the Moreno et al. patent is directed to inserting extra code to perform an "interference" test.

Appellant has disclosed exemplary embodiments for modifying serial dependencies in a procedure. As exemplified in FIG. 10A, it is possible to remove a dependency edge from 1006 to 1008 and replace it with a new dependency edge from 1002 to 1008. A dashed arrow from 1002 to 1008 represents this new dependency edge. Upon removal of the edge, or dependency, from 1006 to 1008, a

compiler 58 can move load 1008 out of the repetitive loop, as shown in FIG. 10B (e.g., specification at page 20, lines 25-33). Such a movement of memory operation as shown in FIG. 10B can provide enhanced and efficient machine code compared to the prior depiction of FIG. 10A because load 1008 is executed only once in FIG. 10B rather than on a repetitive basis as in FIG. 10A (e.g., specification at page 21, lines 1-3). The exemplary method for modifying serial dependencies in a procedure uniquely defines an appropriate dataflow algorithm capable of moving data dependencies out of loops to provide enhanced and efficient machine code.

The foregoing features are broadly encompassed by claim 1 which recites, among other features, a method for modifying serial dependencies in a procedure, including moving a second memory operation to a new position in a graph representation that is closer to a first memory operation, wherein the moving step includes (i) removing one or more of serial dependencies in an initial set of serial dependencies that is associated with the second memory operation and (ii) creating a new serial dependency between the first memory operation and the second memory operation.

The documents relied upon by the Examiner, regardless of whether they are considered alone or in the combination discussed by the Examiner, fail to teach or suggest features of claims 1, 13 and 25. The documents relied upon by the Examiner fail to teach or suggest a method for modifying serial dependencies in a procedure which defines an appropriate dataflow algorithm to move data dependencies out of loops.

The Examiner admits at page 7 of the final Office Action that "Moreno does not expressly disclose the limitations wherein the moving step includes (i) removing one or more of serial dependencies in an initial set of serial dependencies that is associated with the second memory operation and (ii) creating a new serial dependency between said first memory operation and said second memory operation." The Moreno et al. patent teaches inserting extra code to perform an "interference" test, but the Moreno et al. patent does not remove and create serial dependencies as a part of reconfiguring a relocatable memory operation. The Moreno et al. patent does not teach or suggest a method for modifying serial dependencies in a procedure, including moving a second memory operation to a

new position in a graph representation that is closer to a first memory operation, wherein the moving step includes (i) removing one or more of serial dependencies in an initial set of serial dependencies that is associated with the second memory operation and (ii) creating a new serial dependency between the first memory operation and the second memory operation, as recited in claim 1, and as similarly recited in claims 13 and 25.

The Muthukumar et al. patent does not cure the deficiencies of the Moreno et al. patent. The Muthukumar et al. patent discloses testing of serial dependencies by facilitating speculative substitution for comparison of execution results. As shown in FIG. 3, the Muthukumar et al. patent discloses a method 300 for implementing compare speculation in software-pipelined loops by modifying a data dependence graph to enable compare speculation. Specifically, the Muthukumar et al. patent discloses replacing a loop-carried-edge 310 from the data dependence graph with a speculative loop-carried edge 320 for speculative (test) execution and comparison (col. 4, lines 37-44). However, the control speculative execution as taught by the Muthukumar et al. patent does not teach or suggest removing and creating serial dependencies as a part of reconfiguring a relocatable memory operation. The Muthukumar et al. patent fails to teach or suggest a method for modifying serial dependencies in a procedure, including moving a second memory operation to a new position in a graph representation that is closer to a first memory operation, wherein the moving step includes (i) removing one or more of serial dependencies in an initial set of serial dependencies that is associated with the second memory operation and (ii) creating a new serial dependency between the first memory operation and the second memory operation, as recited in claim 1, and as similarly recited in claims 13 and 25..

The Bharadwaj patent does not cure the deficiencies of the Moreno et al. patent and the Muthukumar et al. patent. The Bharadwaj patent discloses that dependency path vectors can be used to "determine if particular code motion or code reordering may be performed without altering the meaning of the source code" (col. 6, lines 41-43). However, the Bharadwaj patent does not teach or suggest removing and creating serial dependencies as a part of reconfiguring a relocatable memory operation.

The Ruf patent, the '398 Radigan patent, and the '704 Radigan patent do not cure the deficiencies of the Moreno et al. patent, the Muthukumar et al. patent and the Bharadwaj patent, even when considered in the combination relied upon by the Examiner. The Ruf patent was cited for its disclosure of a dependence analysis module 314 and a partitioning algorithm module 316 (col. 9, line 66 to col. 10, line 21); the '398 Radigan patent was cited for its disclosure of an intermediate language representation (e.g., col. 6, lines 56-61); and the '704 Radigan patent was cited for its disclosure of optimizing a loop in a computer program (e.g., col. 6, lines 3-11). The Ruf patent, the '398 Radigan patent, and the '704 Radigan patent do not teach or suggest at least the moving step, including removing one or more of serial dependencies in an initial set of serial dependencies that is associated with the second memory operation and creating a new serial dependency between the first memory operation and the second memory operation, as recited in claim 1, and as similarly recited in claims 13 and 25.

B. Combining The Moreno et al. Patent And The Secondary References Would Not Have Resulted In Modifying Serial Dependencies In a Procedure As Variousy Recited In Claims 1, 13 and 25.

On page 3 of the final Office Action, the Examiner responds to the Appellant's earlier arguments, asserting that "Moreno discloses a method for modifying serial dependencies in a procedure..., including moving a second memory operation to a new position that is closer to a first memory operation. Furthermore, Muthukumar discloses (i) removing a serial dependency and (ii) creating a new serial dependency in a graph representation of a procedure..., so as to improve scheduling flexibility and execution efficiently." Appellant respectfully disagrees with the Examiner.

Even if the references could have been combined in the manner asserted by the Examiner, the combination would not have resulted in the claimed method, product or system for modifying serial dependencies in a procedure. For example, the Moreno et al. patent relates to insertions of extra code for an "interference" test, but this does not teach or suggest a dataflow algorithm capable of removing and creating serial dependencies as a part of reconfiguring a relocatable memory

operation. The Muthukumar et al. patent relates to the use of speculative features of a test hardware for comparison of execution results, but the Muthukumar et al. patent does not eliminate a serial dependence in a dataflow algorithm. Further, the Ruf patent merely discloses a dataflow algorithm having interdependences among program types, but the Ruf patent does not teach or suggest a dataflow algorithm that analyzes and stores dependences among actual memory ops of a program. The Bharadwaj patent does not have any dataflow algorithm relating to store dependency information for a set of instructions having reconfigurable control flows, such as loops. Accordingly, combining features from the Moreno et al. patent, the Muthukumar et al. patent, the Bharadwaj patent and the Ruf patent, in the manner asserted by the Examiner, would not have resulted in a dataflow algorithm for modifying serial dependencies in a procedure, including moving a second memory operation to a new position in a graph representation that is closer to a first memory operation, as variously recited in claims 1, 13 and 25.

Further, the combination of references as relied upon by the Examiner would not have taught or suggested a dataflow algorithm capable of removing and creating serial dependencies as a part of reconfiguring a relocatable memory operation, as variously recited in claims 1, 13 and 25. The remaining dependent claims recite additional advantageous features which further distinguish over the documents relied upon by the Examiner.

For at least the reasons discussed above, it is thus respectfully submitted that the claims are in condition for allowance.

VIII. Claims Appendix

See attached Claims Appendix for a copy of the claims involved in the appeal.

IX. Evidence Appendix

NONE.

X. Related Proceedings Appendix

NONE.

XI. Conclusion

For the reasons discussed above, Appellant respectfully submits that the Examiner's decision finally rejecting Claims 1-39 should be reversed.


Respectfully submitted,

Buchanan Ingersoll & Rooney PC

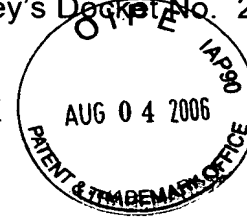
Date

August 4, 2006

By:


78,360
Patrick C. Keane
Registration No. 32858

P.O. Box 1404
Alexandria, VA 22313-1404
703 836 6620



VIII. CLAIMS APPENDIX

The Appealed Claims

1. A method for modifying serial dependencies in a procedure, said method comprising:

building a graph representation of said procedure, said graph representation having an origin and including a unique position, relative to said origin, for each memory operation in said procedure;

designating a location type for each memory operation in said graph representation; each said location type based on a characteristic of said corresponding memory operation;

generating a summary for each memory operation in the graph representation that indicates for each location type used in the procedure the closest preceding memory operation to affect that location type;

identifying a first memory operation having the same location type as a second memory operation; wherein said first memory operation is positioned closer to said origin than said second memory operation, and said graph representation does not include any additional memory operations of the same location type between the first and second memory operations; and

moving said second memory operation to a new position in said graph representation that is closer to said first memory operation, wherein the moving step includes (i) removing one or more of the serial dependencies in an initial set of serial dependencies that is associated with said second memory operation and (ii) creating a new serial dependency between said first memory operation and said second memory operation.

2. The method of claim 1 wherein

the building step includes the step of assigning an initial set of serial dependencies between program operations represented in said graph representation.

3. The method of claim 1 wherein said graph representation is an intermediate representation.

4. The method of claim 3 wherein said intermediate representation is a static single assignment graph embedded in a control flow graph.

5. The method of claim 1 wherein said moving step results in said second memory operation being advanced in a schedule of machine instructions.

6. The method of claim 1 wherein said first memory operation is positioned before a repetitive loop in said procedure and said second memory operation is within said repetitive loop; said graphic representation including a phi node that corresponds to a loop back position in said repetitive loop;

said designating step further comprising a step of advancing through said repetitive loop in order to determine a location type for each memory operation in said repetitive loop;

wherein

when a memory operation having the same location type as said first and second memory operation exists in said loop, said new position in said graph representation is a position that is serially dependent upon said phi node; and

when a memory operation having the same location type as said first and second memory operation does not exist in said loop, said new position in said graph representation is a position that is not serially dependent on any operation in the loop.

7. The method of claim 1 wherein said first memory operation is a store or an array store and said second memory operation is a load or an array load.

8. The method of claim 1 wherein said procedure includes a calling procedure and said first memory operation is in said calling procedure and said second memory operation is in a called procedure that is called by an operation in said calling procedure.

9. The method of claim 8 wherein said moving step results in a placement of said second memory operation in said calling procedure.

10. The method of claim 1 wherein said location type of each memory operation is selected from the group consisting of a predefined set of base types, a predefined set of array types, object types, and object field types.

11. The method of claim 1 wherein the building step further comprises adding a global store dependency to each operation in said procedure that reads a variable from or stores a variable to memory; the method further comprising:

generating a schedule of machine instructions in accordance with said graph representation, wherein each said machine instruction in said schedule of machine instructions, which corresponds to an operation that reads a variable from or stores a variable to memory, is ordered in accordance with said global store dependency associated with said operation.

12. The method of claim 1 wherein a first operation affects a value of a variable stored in memory and a second operation serially follows said first operation, said building step further comprising adding a global store dependency from said second operation to said first operation; the method further comprising:

generating a schedule of machine instructions in accordance with said graph representation, wherein said machine instructions in said schedule of machine instructions corresponding to said second operation are scheduled after said machine instructions corresponding to said first operation.

13. A computer program product for use in conjunction with a computer system, the computer program product capable of modifying serial dependencies in a procedure, the computer program product comprising a computer readable storage medium and a computer program mechanism embedded therein, comprising:

instructions for building a graph representation of said procedure, said graph representation having an origin and including a unique position, relative to said origin, for each memory operation in said procedure;

instructions for designating a location type for each memory operation in said graph representation; each said location type based on a characteristic of said corresponding memory operation;

instructions for generating a summary for each memory operation in the graph representation that indicates for each location type used in the procedure the closest preceding memory operation to affect that location type;

instructions for identifying a first memory operation having the same location type as a second memory operation; wherein said first memory operation is positioned closer to said origin than said second memory operation, and said graph representation does not include any additional memory operations of the same location type between said first and second memory operations; and

instructions for moving the second memory operation to a new position in said graph representation that is closer to said first memory operation, wherein the instructions for moving include (i) instructions for removing one or more of the serial dependencies in an initial set of serial dependencies that is associated with said second memory operation and (ii) creating a new serial dependency between said first memory operation and said second memory operation.

14. The computer program product of claim 13 wherein:
the instructions for building include instructions for assigning an initial set of serial dependencies between program operations represented in the graph.

15. The computer program product of claim 13 wherein said graph representation is an intermediate representation.

16. (Original) The computer program product of claim 15 wherein said intermediate representation is a static single assignment graph embedded in a control flow graph.

17. The computer program product of claim 13 wherein said moving step results in said second memory operation being advanced in a schedule of machine instructions.

18. The computer program product of claim 13 wherein said first memory operation is positioned before a repetitive loop in said procedure and said second memory operation is within said repetitive loop; said graphic representation including a phi node that corresponds to a loop back position in said repetitive loop;

said instructions for designating further comprising instructions for advancing through said repetitive loop in order to determine a location type for each memory operation in said repetitive loop; wherein

when a memory operation having the same location type as said first and second memory operation exists in said loop, said new position in said graph representation is a position that is serially dependent upon said phi node; and

when a memory operation having the same location type as said first and second memory operation does not exist in said loop, said new position in said graph representation is a position that is not serially dependent on any operation in the loop.

19. The computer program product of claim 13 wherein said first memory operation is a store or an array store and said second memory operation is a load or an array load.

20. The computer program product of claim 13 wherein said procedure includes a calling procedure and said first memory operation is in said calling procedure and said second memory operation is in a called procedure that is called by an operation in said calling procedure.

21. The computer program product of claim 20 wherein said moving step results in a placement of said second memory operation in said calling procedure.

22. The computer program product of claim 13 wherein said location type of each memory operation is selected from the group consisting of a predefined set of base types, a predefined set of array types, object types, and object field types.

23. The computer program product of claim 13 wherein the building step further comprises adding a global store dependency to each operation in said procedure that reads a variable from or stores a variable to memory; the computer program mechanism further comprising:

instructions for generating a schedule of machine instructions in accordance with said graph representation, wherein each said machine instruction in said schedule of machine instructions, which corresponds to an operation that reads a variable from or stores a variable to memory, is ordered in accordance with said global store dependency associated with said operation.

24. The computer program product of claim 13 wherein a first operation affects a value of a variable stored in memory and a second operation serially follows said first operation, said building step further comprising adding a global store dependency from said second operation to said first operation; the computer program mechanism further comprising:

instructions for generating a schedule of machine instructions in accordance with said graph representation, wherein said machine instructions in said schedule of machine instructions corresponding to said second operation are scheduled after said machine instructions corresponding to said first operation.

25. A computer system for modifying serial dependencies in a procedure, comprising:

a memory to store instructions and data;

a processor to execute the instructions stored in the memory;

the memory storing:

instructions for building a graph representation of said procedure, said graph representation having an origin and including a unique position, relative to said origin, for each memory operation in said procedure;

instructions for designating a location type for each memory operation in said representation; each said location type based on a characteristic of said corresponding memory operation;

instructions for generating a summary for each memory operation in the graph representation that indicates for each location type used in the procedure the closest preceding memory operation to affect that location type;

instructions for identifying a first memory operation having the same location type as a second memory operation; wherein said first memory operation is positioned closer to said origin than said second memory operation, and said graph representation does not include any additional memory operations of the same location type between the first and second memory operations; and

instructions for moving said second memory operation to a new position in said graph representation that is closer to said first memory operation, wherein the instructions for moving include instructions for removing one or more of the serial dependencies in an initial set of serial dependencies and creating a new serial dependency from the first memory operation to the second memory operation.

26. (Currently Amended) The computer system of claim 25 wherein:
the instructions for building include instructions for assigning an initial set of serial dependencies between program operations represented in the graph.

27. The computer system of claim 25 wherein said graph representation is an intermediate representation.

28. The computer system of claim 27 wherein said intermediate representation is a static single assignment graph embedded in a control flow graph.

29. The computer system of claim 25 wherein said instructions for moving results in said second memory operation being advanced in a schedule of machine instructions.

30. The computer system of claim 25 wherein said first memory operation is positioned before a repetitive loop in said procedure and said second memory operation is within said repetitive loop; said graphic representation including a phi node that corresponds to a loop back position in said repetitive loop;

said instructions for designating further comprising instructions for advancing through said repetitive loop in order to determine a location type for each memory operation in said repetitive loop; wherein

when a memory operation having the same location type as said first and second memory operation exists in said loop, said new position in said graph representation is a position that is serially dependent upon said phi node; and

when a memory operation having the same location type as said first and second memory operation does not exist in said loop, said new position in said graph representation is a position that is not serially dependent on any operation in the loop.

31. The computer system of claim 25 wherein said first memory operation is a store or an array store and said second memory operation is a load or an array load.

32. The computer system of claim 25 wherein said procedure includes a calling procedure and said first memory operation is in said calling procedure and said second memory operation is in a called procedure that is called by an operation in said calling procedure.

33. The computer system of claim 25 wherein said instructions for moving results in a placement of said second memory operation in said calling procedure.

34. The computer system of claim 25 wherein said location type of each memory operation is selected from the group consisting of a predefined set of base types, a predefined set of array types, object types, and object field types.

35. The computer system of claim 25 wherein the instructions for building further comprises adding a global store dependency to each operation in said procedure that reads a variable from or stores a variable to memory; the memory further storing:

instructions for generating a schedule of machine instructions in accordance with said graph representation, wherein each said machine instruction in said schedule of machine instructions, which corresponds to an operation that reads a variable from or stores a variable to memory, is ordered in accordance with said global store dependency associated with said operation.

36. The computer system of claim 25 wherein a first operation affects a value of a variable stored in memory and a second operation serially follows said first operation, said building step further comprising adding a global store dependency from said second operation to said first operation; the memory further storing:

instructions for generating a schedule of machine instructions in accordance with said graph representation, wherein said machine instructions in said schedule of machine instructions corresponding to said second operation are scheduled after said machine instructions corresponding to said first operation.

37. The method of claim 1 wherein when no known preceding memory operation in the procedure affects a location type, said summary indicates the origin for that location type.

38. The computer program product of claim 13 wherein when no known preceding memory operation in the procedure affects a location type, said summary indicates the origin for that location type.

39. The computer system of claim 25 wherein when no known preceding memory operation in the procedure affects a location type, said summary indicates the origin for that location type.

IX. EVIDENCE APPENDIX

NONE

X. RELATED PROCEEDINGS APPENDIX

NONE